Amendments to the Specification:

Please replace paragraph [0001] with the following amended paragraph:

[0001] This application claims priority to U.S. Provisional Application Serial No. 60/400,391 titled "JSM Protection," filed July 31, 2002, incorporated herein by reference. This application also claims priority to EPO Application No. 03291919.3, filed July 30, 2003 and entitled "Micro-Sequence Execution In A Processor," incorporated herein by reference. This application also may contain subject matter that may relate to the following commonly assigned co-pending applications incorporated herein by reference: "System And Method To Automatically Stack And Unstack Java Local Variables," Serial No. [[ ]]10/632,228, filed July 31, 2003~~, Attorney Docket No. TI-35422 (1962-05401)~~; "Memory Management Of Local Variables," Serial No. [[ ]]10/632,067, filed July 31, 2003~~, Attorney Docket No. TI-35423 (1962-05402)~~; "Memory Management Of Local Variables Upon A Change Of Context," Serial No. [[ ]]10/632,076, filed July 31, 2003~~, Attorney Docket No. TI-35424 (1962-05403)~~; "A Processor With A Split Stack," Serial No. [[ ]]10/632,079, filed July 31, 2003~~, Attorney Docket No. TI-35425(1962-05404)~~; "Using IMPDEP2 For System Commands Related To Java Accelerator Hardware," Serial No. [[ ]]10/632,069, filed July 31, 2003~~, Attorney Docket No. TI-35426 (1962-05405)~~; "Test With Immediate And Skip Processor Instruction," Serial No. [[ ]]10/632,214, filed July 31, 2003~~, Attorney Docket No. TI-35427 (1962-05406)~~; "Test And Skip Processor Instruction Having At Least One Register Operand," Serial No. [[ ]]10/632,084, filed July 31, 2003~~, Attorney Docket No. TI-35248 (1962-05407)~~; "Synchronizing Stack Storage," Serial No.

[[ ]]10/631,422, filed July 31, 2003~~, Attorney Docket No. TI-35429 (1962-05408)~~; "Methods And Apparatuses For Managing Memory," Serial No. [[ ]]10/631,252, filed July 31, 2003~~, Attorney Docket No. TI-35430 (1962-05409)~~; "Write Back Policy For Memory," Serial No. [[ ]]10/631,185, filed July 31, 2003~~, Attorney Docket No. TI-35431 (1962-05410)~~; "Methods And Apparatuses For Managing Memory," Serial No. [[ ]]10/631,205, filed July 31, 2003~~, Attorney Docket No. TI-35432 (1962-05411)~~; "Mixed Stack-Based RISC Processor," Serial No. [[ ]]10/631,308, filed July 31, 2003~~, Attorney Docket No. TI-35433 (1962-05412)~~; "Processor That Accommodates Multiple Instruction Sets And Multiple Decode Modes," Serial No. [[ ]]10/631,246, filed July 31, 2003~~, Attorney Docket No. TI-35434 (1962-05413)~~; "System To Dispatch Several Instructions On Available Hardware Resources," Serial No. [[ ]]10/631,585, filed July 31, 2003~~, Attorney Docket No. TI-35444 (1962-05414)~~; "Program Counter Adjustment Based On The Detection Of An Instruction Prefix," Serial No. [[ ]]10/632,222, filed July 31, 2003~~, Attorney Docket No. TI-35452 (1962-05416)~~; "Reformat Logic To Translate Between A Virtual Address And A Compressed Physical Address," Serial No. [[ ]]10/632,215, filed July 31, 2003~~, Attorney Docket No. TI-35460 (1962-05417)~~; "Synchronization Of Processor States," Serial No. [[ ]]10/632,024, filed July 31, 2003~~, Attorney Docket No. TI-35461 (1962-05418)~~; "Conditional Garbage Based On Monitoring To Improve Real Time Performance," Serial No. [[ ]]10/631,195, filed July 31, 2003~~, Attorney Docket No. TI-35485 (1962-05419)~~; "Inter-Processor Control," Serial No. [[ ]]10/631,120, filed July 31, 2003~~, Attorney Docket No. TI-35486 (1962-05420)~~; "Cache Coherency In A

Multi-Processor System," Serial No. [[_____]]10/632,229, filed July 31, 2003~~, Attorney Docket No. TI-35637 (1962-05421)~~; "Concurrent Task Execution In A Multi-Processor, Single Operating System Environment," Serial No. [[_____]]10/632,077, filed July 31, 2003~~, Attorney Docket No. TI-35638 (1962-05422)~~; and "A Multi-Processor Computing System Having A Java Stack Machine And A RISC-Based Processor," Serial No. [[_____]]10/631,939, filed July 31, 2003~~, Attorney Docket No. TI-35710 (1962-05423)~~.

Please replace paragraph [0014] with the following amended paragraph:

[0014] Referring now to Figure 1, a system 100 is shown in accordance with a preferred embodiment of the invention. As shown, the system includes at least two processors 102 and 104. Processor 102 is referred to for purposes of this disclosure as a Java Stack Machine ("JSM") and processor 104 may be referred to as a Main Processor Unit ("MPU"). System 100 may also include memory 106 coupled to both the JSM 102 and MPU 104 and thus accessible by both processors. At least a portion of the memory 106 may be shared by both processors meaning that both processors may access the same shared memory locations. Further, if desired, a portion of the memory 106 may be designated as private to one processor or the other. System 100 also includes a Java Virtual Machine ("JVM") 108, compiler 110, and a display 114. The ~~MPU 104~~JSM 102 preferably includes an interface to one or more input/output ("I/O") devices such as a keypad to permit a user to control various aspects of the system 100. In addition, data streams may be received from the I/O space into the JSM 102 to be processed by the JSM 102. Other components (not specifically shown) may be included as desired for various applications.

Please replace paragraph [0023] with the following amended paragraph:

[0023] The data storage 122 generally comprises data cache ("D-cache") 124 and data random access memory ("D-RAMset") 126. Reference may be made to copending

applications U.S. Serial ~~nos~~Nos. 09/591,537 filed June 9, 2000~~ (atty docket TI-29884)~~, 09/591,656 filed June 9, 2000~~ (atty docket TI-29960)~~, and 09/932,794 filed August 17, 2001~~ (atty docket TI-31351)~~, all of which are incorporated herein by reference. The stack (excluding the micro-stack 146), arrays and non-critical data may be stored in the D-cache 124, while Java local variables, critical data and non-Java variables (e.g., C, C++) may be stored in D-RAM 126. The instruction storage 130 may comprise instruction RAM ("I-RAM") 132 and instruction cache ("I-cache") 134. The I-RAMset 132 may be used for "complex" micro-sequenced Bytecodes or micro-sequences or predetermined sequences of code, as will be described below. The I-cache 134 may be used to store other types of Java bytecode and mixed Java/C-ISA instructions.

Please replace paragraph [0024] with the following amended paragraph:

[0024] As noted above, the C-ISA instructions generally complement the standard Java Bytecodes. For example, the compiler 110 may scan a series of Java Bytecodes 112 and replace a complex Bytecode with a micro-sequence as explained previously. The micro-sequence may be created to optimize the function(s) performed by the replaced complex Bytecodes.

Please replace paragraph [0026] with the following amended paragraph:

[0026] The micro-sequence vector table 162 may be implemented in the decode logic 152 or as separate logic in the JSM 102. The micro-sequence vector table 162 preferably includes a plurality of entries 164. The entries 164 may include one entry for each Bytecode that the JSM may receive. For example, if there are a total of 256 Bytecodes, the micro-sequence vector table 162 preferably comprises at least 256 entries. Each entry 164 preferably includes at least two fields—a field 166 and an associated field 168. The associated ~~F~~field 168 may comprise a single bit that indicates whether the instruction 170 is to be directly executed or whether the ~~associated~~ field 166 contains a reference to a micro-sequence. For example, ~~a bit~~the associated field 168 comprising a bit having a value of "0" may indicate the field 166 is invalid and thus, the corresponding Bytecode from instructions 170 is directly executable by the JSM 102. ~~Bit~~The associated field 168

comprising a bit having a value of "1" may indicate that the ~~associated~~ field 166 contains a reference to a micro-sequence.

Please replace paragraph [0027] with the following amended paragraph:

[0027] If the ~~bit~~associated field 168 indicates that the ~~associated~~ field 166 includes a reference to a micro-sequence, the reference may comprise the full starting address in instruction storage 130 of the micro-sequence or a part of the starting address that can be concatenated with a base address that may be programmable in the JSM. In the former case, field 166 may provide as many address bits as are required to access the full memory space. In the latter case, a register within the JSM registers 140, or preferably within a JSM configuration register accessible through an indirect addressing mechanism using the IRI register, is programmed to hold the base address and the vector table 162 may supply only the offset to access the start of the micro-sequence. Most or all JSM internal registers 140 and any other registers preferably are accessible by the ~~main processor unit~~MPU 104 and, therefore, may be modified by the JVM as necessary. Although not required, this latter addressing technique may be preferred to reduce the number of bits needed within field 166. At least a portion 180 of the instruction 130 may be allocated for storage of micro-sequences and thus the starting address may point to a location in micro-sequence storage 130 at which a particular micro-sequence can be found. The portion 180 may be implemented in I-RAM 132 shown above in Figure 2.

Please replace paragraph [0028] with the following amended paragraph:

[0028] Although the micro-sequence vector table 162 may be loaded and modified in accordance with a variety of techniques, the following discussion includes a preferred technique. The vector table 162 preferably comprises ~~a~~ JSM resources that ~~is~~are addressable via register R14 functioning as an indirect register index ("IRI") register as mentioned above. A single entry 164 or a block of entries within the vector table 162 may be loaded by information from the data cache 124 (Figure 2). When loading multiple entries (e.g., all of the entries 164) in the table 162, a repeat loop of instructions may be executed. Prior to executing the repeat loop, a register (e.g., R0) preferably is loaded

with the starting address of the block of memory containing the data to load into the table. Another register (e.g., R1) preferably is loaded with the size of the block to load into the table. Register R14 is loaded with the value that corresponds to the first entry in the vector table that is to be updated/loaded. An "I" bit in the status register R15 preferably is set to indicate that the register R14 is intended for use as an IRI register. Otherwise, the "I" bit specifies that register R14 is to be used as a general purpose register.

Please replace paragraph [0030] with the following amended paragraph:

[0030] In operation, the decode logic 152 uses a Bytecode from instructions 170 as an index into micro-sequence vector table 162. Once the decode logic 152 locates the indexed entry 164, the decode logic 152 examines the associated ~~bit~~field 168 to determine whether the Bytecode is to be replaced by a micro-sequence. If the ~~bit~~associated field 168 indicates that the Bytecode can be directly processed and executed by the JSM, then the instruction is so executed. If, however, the ~~bit~~associated field 168 indicates that the Bytecode is to be replaced by a micro-sequence, then the decode logic 152 preferably changes this instruction into a "NOP" and sets the micro-sequence-active bit (described above) in the status register R15. In another embodiment, the JSM's pipe may be stalled to fetch and replace this micro-sequenced instruction by the first instruction of the micro-sequence. Changing the micro-sequenced Bytecode into a NOP while fetching the first instruction of the micro-sequence permits the JSM to process multi-cycle instructions that are further advanced in the pipe without additional latency. The micro-sequence-active bit may be set at any suitable time such as when the micro-sequence enters the JSM execution stage (not specifically shown).

Please replace paragraph [0033] with the following amended paragraph:

[0033] As discussed above, one or more Bytecodes may be replaced with a micro-sequence or group of other instructions. Such replacement instructions may comprise any suitable instructions for the particular application and situation at hand. At least some such suitable instructions are disclosed in co-pending application entitled "Mixed Stack-

Based RISC Processor," (~~atty docket~~Serial No.~~ no.~~ ~~TI-3543~~10/631,308), incorporated herein by reference.

Please replace paragraph [0010] with the following amended paragraph:

[0010] Figure 4 illustrates the preferred operation of the JSM to include "micro-sequences"; ~~and~~

Please replace paragraph [0011] with the following amended paragraph:

[0011] Figure 5 depicts an exemplary embodiment of the system described herein[[.]];

Please add the following new paragraph [0011.1]:

[0011.1] Figure 6A illustrates a method in accordance with at least some embodiments; and

Please add the following new paragraph [0011.2]:

[0011.2] Figure 6B illustrates a method in accordance with alternative embodiments.

Please add the following new paragraph [0034.1]:

[0034.1] Figure 6A illustrates a method in accordance with at least some embodiments. In this exemplary method, the information (*e.g.*, one or more indicators) is provided to the vector table from a block of memory accessible to the processor (*e.g.*, the data cache) by an indirect addressing mode (*e.g.*, via the indirect register index) used in a repeat loop comprising at least one instruction. Specifically, the method starts (block 620) and moves to configuring a register as an indirect register index (IRI) (block 622). Thereafter, an individual entry within the vector table may be addressed via the IRI (block 624). The individually addressed entry may then be loaded with information from the data cache (block 626), and the method ends (block 628).

Please add the following new paragraph [0034.2]:

[0034.2] Figure 6B illustrates a method in accordance with alternative embodiments. The method starts (block 630) and moves to configuring a register as an indirect register index (IRI) (block 632). Thereafter, multiple entries (*i.e.*, a block of entries) within the vector table may be addressed via the IRI (block 634). The entries may then be loaded with information from the data cache utilizing a repeat loop of instructions comprising at least one instruction (block 636), but preferably comprising two instructions. Finally, the method ends (block 638).